NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

A VISUAL DEVELOPMENT METHODOLOGY FOR THE DEPARTMENT OF DEFENSE

by

John N. Hodges

September, 1995

Thesis Advisors:	James C. Emery
	Kishore Sengupta

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September, 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE: A VISUAL DEVELOPMENT METHODOLOGY FOR THE DEPARTMENT OF DEFENSE	5. FUNDING NUMBERS
6. AUTHOR(S) Hodges, John N.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Currently, the Department of Defense (DoD) proscibes and encourages the use of Structured Analysis and Design for the development of administrative information systems. However, this methodology has proven to be ineffective in building applications for the DoD. The process has been marked by late and over-budget projects.

A new class of development tools, called visual development tools, promise tremendous productivity and quality gains to developers. These tools provide easy access to database management systems and enable developers to quickly build the normally code-intensive user interface portions of Administrative Information Systems.

Using these tools, however, requires a new methodology to properly implement them and take advantage of their capabilities. This thesis present a new development methodology that incorporates the use of visual tools. Using visual tools to their full advantage, the new methodology calls for a tightly iterative development process that takes into account users inputs thoughout the entire development cycle. Though unproven and not without problems, the visual development methodology is a starting point for incorporating the use of visual tools into the DoD's development process.

14. SUBJECT TERMS Visual Development Tools, Rapid Application Development, Development Methodologies, Software Engineering			15. NUMBER OF PAGES 72
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

A VISUAL DEVELOPMENT METHODOLOGY FOR THE DEPARTMENT OF DEFENSE

John N. Hodges
Lieutenant, United States Navy
B.A., Carleton College, 1984

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 1995

Author: [illegible]
John N. Hodges

Approved by: [illegible]
James C. Emery, Thesis Advisor

[illegible]
Kishore Sengupta, Second Reader

[illegible]
Reuben Harris, Chairman
Department of Systems Management

ABSTRACT

Currently, the Department of Defense (DoD) proscibes and encourages the use of Structured Analysis and Design for the development of administrative information systems. However, this methodology has proven to be ineffective in building applications for the DoD. The process has been marked by late and over-budget projects.

A new class of development tools, called visual development tools, promise tremendous productivity and quality gains to developers. These tools provide easy access to database management systems and enable developers to quickly build the normally code-intensive user interface portions of Administrative Information Systems.

Using these tools, however, requires a new methodology to properly implement them and take advantage of their capabilities. This thesis present a new development methodology that incorporates the use of visual tools. Using visual tools to their full advantage, the new methodology calls for a tightly iterative development process that takes into account users inputs thoughout the entire development cycle. Though unproven and not without problems, the visual development methodology is a starting point for incorporating the use of visual tools into the DoD's development process.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

The Department of Defense (DoD) is both the largest consumer and the largest producer of software applications in the world. As for any organization, it is imperative that the DoD be efficient in implementing software applications. The DoD must get the most value for its dollar and develop the most cost-effective and productive methods for developing and using the software applications it procures. Currently, the DoD spends somewhere between six and ten billion dollars per year on software development and support. The fact that no firm number can be given for this amount is indicative of the troubles that the DoD is having in managing its development programs. Application projects frequently are delivered late and over budget, if they are delivered at all. When a system is delivered, it often does not meet the original project specifications, or if it does, the application turns out to be something that the user no longer wants or needs. Simply put, the DoD has very little success in designing and implementing software applications. Clearly something needs to be done to improve the application development process in the DoD.

There are currently a number of efforts under way within the DoD to improve the productivity of the development process as well as the quality of the applications actually produced. The DoD is looking into object-oriented design and development and the use of Integrated Computer-aided Software Engineering tools. In addition, it is undertaking a number of projects -- the CIM Initiative, the TAFIM/TRM program, the DISA/CIM Reuse Program, the Major Automated Information System Review Council, etc. (Diskin 1993), -- which are designed to improve the development process and save money. All of them propose gains and improvements that can only be described as modest.

While all of these programs purport to have lofty goals and are designed to improve a system that virtually everyone agrees is badly broken, for the most part, these programs are merely pavement over cow paths (Hammer 1990) They do not really address the fundamental issue at hand: that the methodology used by the DoD to develop automated information systems is deeply flawed and needs radical change in order to produce the drastic improvements necessary. The system has literally become as bad and as flawed as it can be -- to the point where only radical change can bring about the drastic improvement needed to make the system merely tolerable. It is only through a dramatic shift in thinking about the process of developing software and a willingness to embrace new technologies that the DoD will be able to provide a reasonable return on the investment it makes in taxpayer dollars spent on the development of automated information systems (AIS).

B. SCOPE OF THIS THESIS

This thesis will examine the potential use of visual development tools by the DoD. It will first examine the current development tools and methodologies used by the DoD. It will examine the definition of visual tools and their potential advantages and weaknesses. It will then provide a development methodology for use with visual tools and will examine whether this methodology can be more productive and reduce development and maintenance costs while providing more robust applications. In examining visual tools, this thesis will examine the use of object-oriented programming methods and the potential advantages of graphical user interfaces, both for the developers and for the end users.

Visual tools are being examined because they provide the possibility of greatly enhancing the productivity of programmers and the quality of programs that they produce. Through the generation of code and the easy development of user interfaces, visual tools provide a powerful and useful new way to build applications. No tool can be productive without a useful methodology, however, and this thesis will attempt to put forth a methodology that takes advantage of and brings out the power of visual tools.

This thesis will cover only the development of Automated Information Systems (AIS). Automated Information Systems can be defined as software applications that are data intensive and involve transaction processing (Diskin 1993). AIS applications are user interface intensive, as they generally involve numerous inputs and reports. Thus, the development of applications with quality user interfaces is paramount to improving maintenance costs and user productivity. More and more, AIS applications are following the client/server architecture, and AIS applications are beginning to fall into the client/server model. This thesis will not specifically discuss the client/server architecture, but will assume that the development methodology for AIS and client/server applications will be compatible.

C. DOD FAILURES

The track record of the DoD in procuring software applications is marked with failure. Many software projects undertaken by the DoD exceed their estimated costs, frequently by enormous amounts. A house sub-committee report cited a program that was to develop a standard financial accounting system for the Navy's fourteen research and development installations that was originally estimated to total six million dollars. Eventually, upon cancellation of the program, the total life cycle costs had swollen to over $800 million. (House Report 101-382) Such enormous cost overruns are not uncommon.

Government documents cite numerous such cost problems. The following are just two examples of a common problem:

- The Naval Aviation Logistics Command Management Information System (NALCOMIS) was initiated in 1977 to automate record keeping and reporting requirements on aircraft repair, maintenance, and supply activities in the US Navy and Marine Corps. It was expected to reduce administrative costs and inventory loss, improve turnaround time and productivity, and provide greater visibility for the military's assets. The program went through numerous setbacks and performance problems, and in 1985 a cost-plus-award-fee contract was awarded to Arthur Anderson Consulting to do a software redesign on the system. By 1988, the Navy estimated that it had spent $233 million on development and implementation at only 37 out of a potential 500 sites. Divided into three phases, development of phase III was suspended in June 1987 so that funding could be transferred to Phase II to correct problems in that phase. By 1991, total life cycle cost estimates had increased to $1.4 billion. Today, the program continues to be incomplete and grossly over budget. (GAO/IMTEC-89-21FS)

- In 1990, the Air Force Personnel Concept III was scheduled to be deployed even though the system was only partially developed and tested, had not yet passed significant testing milestones, was based on a hardware design selected without

> proper requirements analysis, and was justified on unsupported claims of cost savings. Originally estimated to cost $200 million, the project's cost was put at $550 million at the time of the desired early deployment. The GAO doubted whether the system would function at all, and if it did, whether it would return value anywhere near the cost of building it. (GAO/IMTEC-90-22)

In addition to cost overruns, projects frequently are delivered extremely late, with functionality below that of the original specifications. Project specifications go unmet and users end up with applications that are neither useful nor functional. This naturally raises the question of whether the large sums of money spent on the systems were justified in the first place and whether there was any measurable return on investment. Frequently, to avoid cost overruns, contractors will limit the functionality of the project in an attempt to meet the target budget. Further, by delivering applications with reduced functionality, the cost savings estimated as a benefit of the new project are often not fully realized, increasing even more the overall cost -- both actual costs and opportunity costs of the project -- to the DOD. (House Report 101-382 1989)

For example, the GAO reports (GAO/IMTEC-89-29) that several Air Force systems were delivered with less than full functionality, and thus the overall benefits of the systems and the return on the investment in them were seriously questioned. The Air Force Stock Control and Distribution System was originally supposed to replace 23 existing systems at a cost of approximately $202 million. Instead, only thirteen system were replaced and the project was $21 million over budget. The GAO believed that the benefits received from the limited project did not justify the expense. The Air Force estimated that the project would bring about a cost savings of over $3.1 billion, but was only able to identify cost savings of $193 million.

D. WHY THE ISSUE IS CRITICAL

With the huge amounts of money being spent on systems that offer little in the way of return, the issue of software development is critical to the DoD. Software itself is growing in its importance to the DoD, particularly in areas not normally considered mission-critical such as administrative information systems. As seen above, these systems can become very expensive very fast under the current DoD methodology. In the current environment of shrinking budgets, technology becomes more important to the DoD's success, it must extract the most value for the dollar from its IT budget. It simply cannot afford to continue on its current course. Any inefficiencies in this area can greatly affect the DoD's ability to carry out its mission.

Perhaps the largest cost involved with the substandard performance of DoD software development is the opportunity cost of working with poor software. By not providing current, state-of-the-art software to its users, the DoD pays an opportunity cost in lost productivity, both in terms of end-user productivity and in terms of overall bureaucratic efficiency. By leaving inefficient processes in place -- processes that could be re-engineered and automated using Information Technology -- the DoD misses the cost savings of an efficient, productivity-enhancing software solutions. This benefit is potentially huge, but harder to quantify.

It is clear that the issue of software development costs is a critical area for the DoD. The issue of cost overruns is important, but perhaps even more important is the issue of a loss of functionality. The high costs incurred by the DoD would be more tolerable if the expenditure resulted in complete, useful, and functional systems. The fact that the DoD is not able to deliver working systems is more disturbing than the huge amounts of money being wasted. The mission of the DoD is too important for it to not have functional, working information systems.

E. OVERVIEW

Chapter II discusses structured analysis and design, the current methodology prevalent in DoD development. It will discuss the processes used and some inherent weakness to structured design.

Chapter III introduces visual development tools by defining them, discussing their use, and outlining their inherent strengths and capabilities. It focuses on the use of the object-oriented programming paradigm and graphical user interfaces in visual development tools and the advantages that these tools bring to developers.

Chapter IV discusses a systematic development methodology that can be used with visual development tools. It outlines the steps taken in using the methodology and discusses its strengths and weaknesses.

Potential problems and weaknesses of the visual development methodology are discussed in Chapter V. In addition, this chapter examines potential implementation strategies and difficulties in instituting the new methodology in the DoD.

Chapter VI discusses a small case study done using the visual design methodology, and attempts to show how the visual design methodology might be applied in a real world situation.

Chapter VII draws conclusions and makes recommendations.

II. STRUCTURED ANALYSIS AND DESIGN IN THE DOD

A. GENERAL DISCUSSION

For many years, structured analysis and design has been the standard methodology used by organizations to build their AIS applications. The advent of third-generation structured programming languages such as COBOL and Ada enabled the development of structured design. Structured languages lend themselves to a methodology that follows the same rules as the language, thus structured design was developed. Much effort has been put into standardizing the process in order to add discipline and repeatability to the development of applications. It is believed that if structured analysis and design (SAD) could only be perfected, then the applications created with it will be on time, under budget, and what users need.

The DoD has proscribed that structured analysis and design be the official method for developing AIS applications within the department. DOD Directive 8120.1 and DOD Instruction 8120.2 outline the required life-cycle management procedures and defines the clear-cut milestones to be followed by system developers. DoD Standard 2167A outlined the waterfall methodology as the official DoD development methodology. It was only recently superseded by Military Standard 498 that describes a more flexible standard for developing software, but which still requires strict milestones and profuse documentation. The DoD further solidifies the use of the methodology by ordering the exclusive use of the programming language Ada as the only acceptable language for programming DoD applications.

In its basic form, structured analysis and design calls for developers first to do a requirements analysis, followed by functional decomposition of the business processes of an organization. After the design specifications are complete, the requirements are "frozen," and changes to the application's design are no longer allowed. Upon completing that, developers create structured charts that are then turned into code and a final product. Once the building of the application begins, the developers are sequestered away from any potential users, keeping them from adding requirements and causing "requirements creep." After the application has been implemented, it is tested, and, once approved, deployed. From there, the application moves into the operation and maintenance phase.

The most classic implementation of SAD has been through the waterfall method, as seen in Figure 1. As implemented by the DOD, the waterfall method is marked by clear cut phases and specific deliverables that are required at certain points along the development

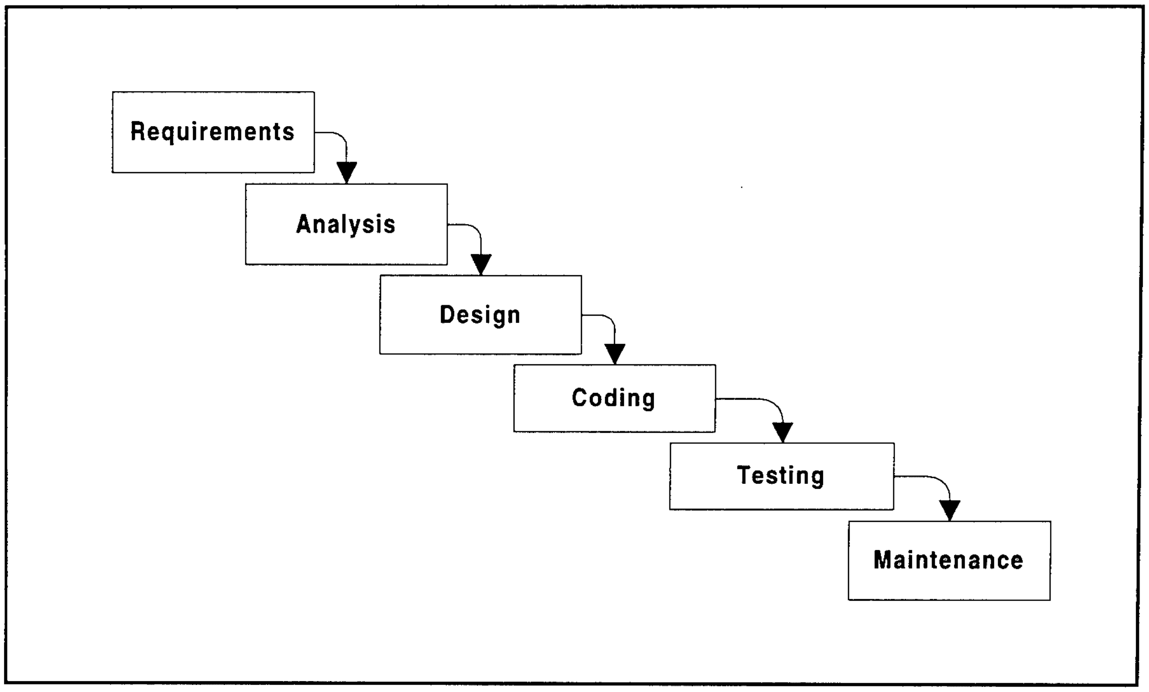

Figure 1 The Waterfall Design Methodology

cycle. Each step is clearly defined and structured to ensure that errors in design and implementation are found as early as possible, and that the process can be closely monitored to ensure that the design is done properly and that the developers comply with the specifications as laid out in earlier steps. Arduous documentation requirements are intended to ensure close control and monitoring of the process. The DOD has even implemented a series of review boards to ensure that the process is done correctly and that the system is built according to specification. Developers are required to very carefully comply with a strict set of guidelines in order to complete the process and finish a project.

Structured analysis and design, like any design methodology, is designed to describe

- Hand coding in a third-generation language
- A "structured programming" development methodology
- An automated project management system
- A database management system
- A mix of on-line and batch applications in the same system
- Development of mostly mainframe applications
- Programming by professional programmers only
- Various automated (but not well integrated) software tools
- A well defined sign-off process for system delivery
- User participation mainly in requirements definition and installation phase

Figure 2 Characteristics of the Traditional Approach to Development (Sprague and McNurlin, page 264-5)

and manage the complexity of the systems it wishes to automate. It attempts to do this by emphasizing discipline, reliability, low error rates, and the efficient use of resources. It injects discipline by establishing the strict standards for documentation and the requirements for clear milestone approval. It stresses higher reliability and fewer errors through inspections. The hope is to catch errors early in the development cycle when they are cheaper to fix. Finally, it is believed that by contolling closely the entire process, managers can better control resource use and allocation, and thus control costs. (Sprague and McNurlin, 1993)

As structured design matured, it became apparent that it was not performing as well as managers had hoped. Building models that provided an abstract look at systems and managed their complexity proved to be more than structured analysis and design could handle. Therefore, great efforts have been made at improving the process. Code reuse, automation through Computer Aided Software Engineering (CASE) tools, and code generation have all been put into practice with the hope that they would improve the productivity of designers and programmers alike. Numerous software packages have been developed that try to automate the process of structured design.

Yet despite all the contol and documentation built into the process, developers are still plagued by cost and schedule overruns and systems that did not meet users needs. Structured analysis and design has proven to be far from perfect, and indeed there are a number of alternatives that are coming to the fore that are proving to be more effective, and commercial developers are beginning to embrace them. The DoD should consider these alternatives as well.

B. TROUBLES WITH STRUCTURED ANALYSIS AND DESIGN

Emery and Zweig (1993) discuss the problems with the current methodology, the low productivity of programmers that use a third-generation language, the size of the teams that are required to build large applications as a result, and the strict controls put in place by project managers to limit the ability of users to request changes to the project specifications. They also note that the current development system almost chronically produces applications that do not meet users current needs. They see major problems with the way the DoD builds software and recommend a new paradigm that promises to dramatically increase productivity, and as a result minimize costs and deliver better, more robust applications.

Structured analysis and design is based on the idea that if developers can get perfect specifications up front, the end result will be a perfect application. It puts a large emphasis on the initial requirements specification. This need was originally driven by the technology of the day. The actual input of a program into the computer was often arduous and error-prone. Computer cards and paper tape did not lend itself to online development. If the program had to be entered into the hardware using such tedious methods, managers were hard pressed to ensure that it ran right the first time, because making changes was not easy.

It is only during the requirements phase that users are consulted as to their needs and desires for the application. Once the requirements specifications are frozen, the process makes further user input very difficult. Developers then strive to design and build the application exactly as specified. Indeed, development managers consider any further inputs or specification changes to be undesirable and detrimental, and often go to great lengths to avoid such changes. The idea of further user input is often met with skepticism and rejection. As a result, if user specifications are changing more rapidly than structured design can react, then users end up with an application that no longer fits their needs.

It is also becoming clear that the business cycle is rapidly shortening and that the development cycle is failing to keep up. Technological changes are occurring at a tremendous rate. Business rules are being changed regularly and the need for new and current applications is pressing developers to produce new products at a faster rate. It is common to hear of large backlogs in an organization's development plans. Commercial developers are finding themselves producing new versions of their products much more frequently. Some companies are even offering 'subscriptions' to products rather than mere updates, allowing them to provide interim and more frequent changes to customers.

In such an environment, methodologies that do not allow for rapid changes in reaction to swiftly altering business rules can no longer provide the products that users need and want. As a result, development cycles that take three, four, or even ten years to complete a project are providing applications that are of no use to end users. This problem is compounded when the development process does not allow for changes and delivers a system that meets requirements that were laid out five years before. The rapid pace of technology and change simply leaves behind projects developed with the current development process. Methodologies that do not embrace change are not going to be able to keep up.

The structured design methodology falls into the category of methodologies that are not keeping up. The numerous tales of DoD projects gone awry illustrate this. Because of the large paperwork requirements, limited programmer productivity, and milestone requirements, managers using this methodology find themselves constantly playing catch up -- a game they cannot win. The contracting process itself certainly plays a role in this failure, but it is the methodology itself that cannot provide the needed responsiveness. Measuring development times in years is no way to provide adequate systems to users who measure business changes in months. As implemented by the DoD, development teams have to spend many man-hours merely complying with the bureaucratic requirements of the process itself, further eroding overall productivity. This type of control hardly lends itself to fast development times and inexpensive projects.

This level of control desired by the DoD can be further illustrated by the Department's strong interest in the Capability Maturity Model. The Capability Maturity Model (CMM) is a standard which measures the capabilities and skill levels of software development organizations. It outlines a set of recommended practices and standardized development techniques that stress measurable, repeatable development practices. It guides organizations in how to control the development process. Organizations can measure their current process maturity and determine methods for improving their development capabilities. It also provides five maturity levels that measure an organization's current skills sets. The entire model stresses repeatable, standard development processes that can be used by organizations like the DoD to measure an organizations ability to produce software (Paulk, et al. 1993) .

The CMM therefore becomes just another layer of bureaucracy within the software development process. By forcing developers to meet a predefined standard through the use of the CMM, the DoD will perpetuate the huge overhead involved in developing software and squelch any innovation that may allow for the desperately needed improvements in software development. Organizations that concentrate on raising to a higher level on the CMM are not concentrating on developing useful applications. No two software projects are alike, and any system that attempts to fit every software project into the same shaped hole cannot produce the type of application that can adapt to a changing domain.

Furthermore, as might be expected, the reward system that builds up around this methodology reflects these constraints. Managers are rewarded only for meeting specifications. Success is defined as completing a project on time, under budget, and having met the specifications as laid out in the requirements document. DoD regulations define as successful those projects that can meet certain criteria at certain points in time. Meeting that specification, particularly if the project is measured in years, virtually guarantees that what is delivered at the end of the process is an application that no longer fits user needs. Measuring success in terms of user satisfaction seems only a small part of the equation. The GAO, while frequently outlining failures as seen above, stresses only costs and schedules. There seems to be very little discussion about whether a project actually does what it needs to do. Such a definition of success clearly motivates managers to avoid changes in the system to be built, as changes mean higher costs and later delivery dates. Development teams are not motivated to produce anything other than what is originally planned for. Changes are met with skepticism and hesitancy and users are all too often left out of the process.

C. ALTERNATIVES

As technology advances, users become more sophisticated and require systems that meet current capabilities. Technology has become more and more pervasive, infiltrating many aspects of life that were previously untouched. It is becoming increasingly difficult to find even common household appliances that do not contain some sort of microprocessor. Many of today's users are very familiar with computers. Computers are common in the workplace and growing common in the home. The level of technical skill of the average user is growing. Users are familiar with applications that employ graphical user interfaces and have come to expect. Such users can be expected to be able to learn a more sophisticated level of application than previously thought. A user who runs Microsoft Windows applications on his or her home machine will find a character-based application used at work to be primitive and non-productive.

Rather than eschewing these inevitable changes, organizations such as the DoD need to embrace them. They need to recognize that the current methods are no longer working.

Rather than fighting a losing battle against a relentless tide, organizations need to embrace a rapidly changing environment. They need to use a methodology than can accept and adapt to changes as they occur. No business process is static; in today's environment changes can occur overnight. Software development methodologies that do not recognize this fact and deal with it will no longer be able to meet the needs of customers and produce the types of software needed.

DoD software projects with huge cost overruns that are grossly over schedule and applications that do not even meet the original specifications, much less the user's current needs, are the result of a failure to recognize and adapt to the current trends in today's business and development environment. It is only through a radical change in the methodology used to develop software that development project managers can begin to meet the needs of users and build the applications they need on time and within budget. The current tools and methodologies being used can no longer be expected to do the job. A methodology that employs visual development tools may provide a solution.

III. VISUAL DEVELOPMENT TOOLS

A. INTRODUCTION

Currently there exists a new generation of software development tools that hold the promise of tremendous productivity gains for developers of AIS applications. Generally called visual development tools, they have become enormously popular in the commercial marketplace. Vendors are frantically competing to produce better and more efficient visual development tools. Visual tools take advantage of the current popular graphical user interfaces to provide programmers with intuitive and easy to use development environments.

Visual tools are a large step up from traditional structured languages. Generally, a visual tool will employ some form of a structured, third-generation procedural language for specifying application logic. However, to a large degree, the developer is shielded from the rudimentary coding level such as writing code for the user interface itself. Visual tools allow for the easy manipulation of the interface design, allowing developers to concentrate on the logic of the application. In addition, visual tools usually conform to a common graphical user interface such as Microsoft Windows, which results in end user applications that are familiar in function and appearance.

As will be discussed, visual tools normally are object-oriented and thus reap the benefits of object-oriented design and programming. Most commercial visual tools on the market today take advantage of a class library of objects that provide immediate code reuse.

B. DEFINITION OF A VISUAL DEVELOPMENT TOOL

Visual development tools offer the potential to greatly reduce the effort and cost involved in building AIS applications. Visual development is a rapidly growing market, with virtually all of the major application vendors scrambling to provide visual development environments for their development tools. As more and more developers see the benefits of developing applications visually, the competition in the marketplace will increase, resulting in more and better tools being offered.

No mere code generators, visual development tools are a revolutionary step in the world of development tools. Visual tools are slowly increasing their share of the development tools market as their benefits are more clearly known. But what exactly is a visual development tool? For the purposes of this thesis, a visual development tool will be defined to have the following attributes:

- It is designed to develop applications for a graphical user interface environment such as Microsoft Windows. It should provide complete access to the Application Programming Interface (API) for its environment.

- It allows developers to build user interfaces by 'painting' on the screen. Such tools are normally form based, allowing developers to select screen elements from a palette and then place those elements on a form or window. Such screen elements might include standard inputs devices such as checkboxes, option buttons, edit boxes, grids, and labels.

- A visual tool should include an object-oriented language that fully supports inheritance, encapsulation, and polymorphism. Preferably, it would use a language that incorporates all the features of current third-generation languages, including pointer access and the ability to address the hardware directly if necessary.

- It should provide a class library that encapsulates the basic screen elements in code, allowing developers to access the capabilities of each visual element seamlessly. It should allow for the development of descendant elements based on the basic elements provided by the environment through objet orientation.

- It should include a database engine (or include connections to one), providing developers with the ability to easily incorporate data-aware control into an application. Such access should be incorporated into the provided class libraries. It should allow for access to both local and remote databases. Access to a variety of DBMS servers through SQL is an important feature. Easy database access is a key feature for any tool used to develop AIS applications.

- Once screen elements are designed, a visual tool should generate skeleton code for the various events that take place as users interact with the application. It should easily facilitate programming for the event-driven environment by allowing the quick development of event handlers, such as keystrokes, mouse clicks, and system messages.

- It should provide some method of producing an easily distributed application, either through a native code compiler or a runtime library that can be included with some form of interpreted code. Visual development tools should provide the ability to install an application on a remote machine without having to include the entire development environment itself.

With a development tool as described above, developers can greatly reduce the amount of code needed to be written by allowing the visual tool to do much of the tedious code involved in designing an interface and handling the various events that take place in a graphical environment.

Examples of such tools might include Delphi for Windows from Borland International or Clarion for Windows from the Top Speed Corporation. These tools provide an object-oriented language with a true compiler. The various Smalltalk implementations provide a true object-oriented visual environment without a true compiler. Products such as Visual Basic from Microsoft and Powerbuilder from Powersoft are visual in nature, but provide interpreted executables and do not include true object-oriented functionality. However, all of these tools can be used to build powerful AIS applications and can provide productivity gains to development teams.

C. USE OF VISUAL DEVELOPMENT TOOLS

Historically, applications have been developed by programmers typing code with some sort of text editor. They then run the compiler on the text file containing their code, which produces an executable file that can then be run and tested. Virtually all of the code -- indeed the entire development process -- is entered and developed manually. Programmers had access to libraries of code that they could either access programmatically or that they could simply cut and paste into the application. All aspects of the program had to be developed by hand, whether it be the user interface, access to a remote database, or the logic of the program itself. Such programming is very labor intensive and subject to numerous errors.

Productivity enhancements were frequently sought through the re-use of code, usually by cutting and pasting previously written code from code libraries or by integrating binary code units into applications. Developers also looked for more productive and efficient code editors -- ones that had better text handling capabilities, macros, drag and drop, etc. Commercial development packages began providing Integrated Development Environments which allowed developers to code, compile, and run applications from a single environment. Such innovations brought about modest and incremental improvements in programmer productivity.

Traditionally, projects developed this way would be measured in person-months or even person-years. Developing large AIS applications with this method takes a large manpower effort. The coding itself is often a large portion of a project, and thus a large portion of the cost. Reducing the amount of time and effort needed to actually code a large project is the goal of every software engineer. Anything that can be done to reduce the traditionally expensive costs involved with programmer labor naturally reduces overall project costs. The currently proscribed methodology for producing software in the DoD leaves a lot of room for improvement in the area of programmer productivity.

D. ADVANTAGES OF VISUAL DEVELOPMENT TOOLS

Visual tools provide a development environment in which developers use graphical pictures of icons to draw user interface items such as list boxes, checkboxes, data entry fields and grids. The basic building blocks of an application are pre-defined, and developers can increase interface development productivity by taking advantage of these pre-built components. The state of each component can be manipulated at design time to create the forms and windows that will make up the application itself. Windows, dialogs, and reports can be graphically manipulated and designed with the use of basic pointing devices. Very little code is actually written in this part of development.

Historically, user interface has been a very large part -- as much as 80 percent in some estimations -- of the total amount of code needed to be written for a new AIS application.

Such code is often painstakingly written, ensuring that user interface elements are properly positioned and that the overall appeal of the application's interface is user-friendly and designed to allow for efficient use by the user. It also involves doing the very time-consuming compile-check-fix-recompile cycle in order to correct the look of the interface. Such coding can cost many man-hours and involve a large portion of a programmer's effort. When using an operating system that does not provide a common user interface, creating one will involve much of this type of work. In this situation, applications will take on many different looks, resulting in each new program encountered by users to appear unfamiliar.

This problem frequently occurs in character-based systems such as MS-DOS and mainframe-based applications. Organizations that use applications that have different user interfaces require unnecessary training expenses as user have to learn a new interface and as well as the new application. Each application requires a certain adjustment period as users have to reorient themselves for each one. Time is lost when users confuse command sets among applications. Some organizations may enforce standards for their interfaces in these situations, but an organization like the DoD, with the wide variety of applications in use throughout, cannot hope to standardize on one interface design without embracing a design incorporated into a graphically-based operating system.

Applications that take advantage of a common graphical user interface have no such problems. A common user interface allows users to learn a basic set of commands and controls, making new applications that conform to such interfaces seems at least somewhat familiar to new users. Users who are accustomed to an interface such as the one provided by Microsoft Windows can easily adapt to a new Windows application. They become familiar with the standard menu commands such as those usually found on the conventional File and Edit menus. Common keystrokes and other interface items are easily incorporated into applications and thus easily used by users who are accustomed to seeing such things in Windows applications. Items such as menus, tool bars, status bars, and dialog boxes are standard fare for users who understand and accept the Windows environment. The same can be said for operating systems such as XWINDOWS, OS/2, or the Macintosh. Such environments make it very easy for developers to provide these familiar interfaces.

Visual development tools provide developers with easy access to these common interface items. By the above definition, a visual tool provides a class library that encapsulates the common interface items such as edit boxes, option buttons, checkbox buttons, and pushbuttons. Visual tools provide for the rapid development of the user interface screens, controls, and dialog boxes. Such interface elements can be quickly developed and at the same time are familiar to users who use the operating environment on a regular basis. By allowing developers to develop the interface by graphical means -- 'painting' or dragging and dropping objects on a form -- and with almost no coding, interface development can become an almost trivial task. Changing, updating, and evaluating the interface can be done as the project is developed with almost no additional effort. Changing a screen becomes merely a task of moving graphical elements around with a pointing device rather than changing written code. Since interface development becomes much more efficient and productive, overall project size shrinks and the number of developers can be lowered, often drastically if the application is interface intensive. If large portions of a development project involve interface development, then it is clear to see that large gains in the productivity of developers building that interface can have great cost benefits to projects that use such tools.

As development team size rises, so do the number of possible lines of communications. This is, in fact, a major factor in the exponential growth of development time as projects get larger. By lowering the number of developers on a project, the number of lines of communication drops rapidly, which increases the productivity of a development team. A project that has two thousand lines of code may only require two developers. This creates only one line of communication. However, a four thousand line project may require three programmers, increasing the number of lines of communication to three. Ten developers, say, result in 55 lines of communication. This number increases exponentially as developers are added to a project. Increased transaction costs that are a result of these lines of communication add to development time because they decrease developer productivity. A developer can only be at his or her peak productivity working on a project alone. Any interaction with other developers is time spent interacting and not developing. The more possible lines of communication means the more communication that will be required. The more communication that is required means lower developer productivity. Lower productivity means higher costs. If visual tools can reduce the number of developers required for any project, then it can lower the overall cost of that project.

E. USING VISUAL TOOLS

It is very tempting to simply take a visual development tool and plug it into one's current methodology. However, visual tools are a revolutionary step in development practice, and to try to place "new wine in old wineskins" simply is not a formula for success. Rather, visual tools alter the entire paradigm of application development, allowing for a completely new way of thinking about the development process. No longer should developers be held to the strict and limiting structured design methodology. Visual tools allow developers to break down barriers enforced by structured design and begin to deal with, and even embrace, its limitations.

Developers should no longer be separated from users, particularly during the development cycle. Rather, users should become a central part of the development process. Instead of freezing requirements and allowing no change as the project is developed, visual tools can allow, even seek out, previously avoided things such as requirements creep and changing needs. As the user's business rules change in a dynamic environment, developers can track and include the changes.

As a result, the line between development and maintenance begins to blur. Developers can actually deliver a working prototype that has the basic functionality of the required system. Users can begin working with this basic design, providing both problem reports and ideas for improvement. Developers can provide updated versions of the application on a timely basis. They can fix problems as they are revealed and provide added functionality, all in the same stage of the development cycle. No longer are developers constrained by a methodology that specifically prohibits such interaction. Users are no longer stuck with a static application that cannot grow and change as their needs grow and change.

As a result, an application may never truly be considered done, but rather could be thought to be continually evolving. Systems will arrive in the hands of users earlier and will meet their specific requirements. This alone will help eliminate two large problem with software development -- incomplete systems delivered late. Users will not end up with an application imposed on them, but will become an integral part of the project's development. This new process can easily result in the savings of many man-months, both on the development side and the user side of an application.

The removal of the strict delineation between development and maintenance should also enhance the quality of applications. With users using applications much sooner in the development cycle, they can provide valuable, real-world testing that cannot be done by the developers themselves. As the project grows incrementally, bugs will be revealed earlier in the development cycle, thus making them easier and cheaper to fix. Also, as the project grows, user inputs will ensure that the project grows in the proper direction and with the proper functionality, thus allowing users to get the work done that needs doing now, not the work that needs doing back when the requirements specifications were frozen.

F. VISUAL TOOLS AND GRAPHICAL USER INTERFACES

A graphical user interface is a key component of a visual tool. Without it, visual tools do not exist. The advantages of graphical user interfaces are many, and certainly the success in the marketplace of Microsoft Windows and IBM's OS/2 testify to their usefulness and productivity benefits. Graphical user interfaces provide benefits in the software development arena in two ways: by increasing the productivity of developers who use them to build applications, and by providing a more productive application to end users who actually employ and use the applications built by developers.

The traditional approach to designing software has normally placed much of its emphasis on the development of the processing tasks required for the application. This functionality is often designed first; the development of a quality user interface only gets secondary treatment. The interface thus often becomes nothing more than an afterthought. Such an approach rarely take into account the human factors considerations needed to create a useful interface. Software design techniques that takes human factors into account -- such as providing a graphical user interface -- produce applications that are easier to use and increase user productivity when employing those applications (Ruston et al. 1991). This same principle can be applied to programmers themselves, who will benefit from the use of a graphical user interface when developing applications.

G. VISUAL TOOLS AND OBJECT-ORIENTED PROGRAMMING

As described above, visual development tools should embrace the object-oriented model and allow developers to build, use, and reuse objects. The object-oriented development paradigm offers developers a number of potential advantages, including code-reuse, easier maintenance of existing code, and a cleaner, less-coupled means of designing applications.

Virtually every visual development tool on the market today claims to be object-oriented, or at the least object-based. More and more developers are embracing this technology as its advantages become clearer. The DoD has done a close examination of the prospects for integrating object-oriented technology into its development process and found it to be very promising.

For a development language or environment to be truly object-oriented, it must have three basic characteristics -- encapsulation, polymorphism, and inheritance. These are discussed in the following section.

1. Object-oriented Architecture

Object-oriented architecture involves a new way of looking at a system. It provides a way of abstracting difficult design concepts in an intuitive model. Rather than the traditional way of viewing as separate entities data and the actions taken upon that data, object-oriented architecture (OOA) is built on the concept of the object. An object is a data structure that contains data as well the functions to act on that data. Objects generally have properties, which describe the state of the object, and methods, which describes an object's behavior and how that object will manipulate its properties and data. In order to be a true object, it must have three basic characteristics -- inheritance, polymorphism, and encapsulation.

Inheritance is the ability of an object to pass on all of its data and functionality on to a subclass. Inheritance allows for the building of a class hierarchy that puts the most abstract common functions of objects at the top, and allows that functionality to be included in sub-classes as new objects are inherited from ancestor classes. Ancestor classes hold data and functions that descendant classes take on when they are inherited from the ancestor. A sub-class is said to inherit the data and functions of a super-class. Inheritance contributes to code reuse by allowing developers to use previously developed objects again, both directly, and by inheriting and enhancing a previously designed object's functionality.

Polymorphism is the ability of different objects to react differently to the same messages. It allows for objects to send messages to other objects without having to know which particular object will respond, nor how it will respond to the message. It allows for developers to take advantage of the similarities in inherited classes. Polymorphism also allows subclasses to respond to the same message in different ways. As a result, an object that sends a message does not need to worry about how a receiving object will react to the message. Polymorphism contributes to code reuse by combining with inheritance and allowing objects to change only part of their behavior while maintaining their main processes.

An object is a collection of all the data and functions that it needs to perform a certain task. The collection of such data and functions together in a class is called encapsulation. Encapsulation provides data-hiding and the ability to provide a black-box functionality. It allows objects to wrap up their behavior in a single entity, allowing complex concepts to be segmented into more easily comprehended chunks. The internal functions of an object are hidden from the users of an object. Encapsulation allows an object to present a specific and defined interface to other objects by protecting data and functions from users. An object only provides information through these interfaces. This allows the internal workings of an object to be altered and enhanced without affecting any other part of the system that deal with the object.

2. Advantages of Object-oriented Architecture

These three characteristics combine to provide full object-oriented functionality to a system. This functionality is proving useful as developers realize that the level of abstraction and the ability to break a system down into its natural components enables them to segment and clearly define their data and data functions. Objects can be defined with clear interfaces, requiring developers using various different objects to know only about how the object's interface works and not about the inner workings of that object. Once the paradigm is understood, object-oriented technology provides developers a more natural way of looking at a system, as objects model the real-world things that make up the system. Objects that define the state and functionality of, say, bank accounts, orders, and people are easily created and understood when the object-oriented technology paradigm is applied to them.

Object-oriented technology makes it easy to derive a system description that is relatively low in coupling and that has high levels of cohesion. By definition, objects are distinct and discreet data types that understand themselves and their own functionality. They are not dependent on other objects to perform actions for them. Since they communicate solely by message passing, objects can and do stand alone. Numerous studies have show how low coupling improves a project's clarity and maintainability, and object-oriented technology allows systems designers to almost eliminate coupling when designing a system. Objects can collaborate with one another through message passing, and since each object has discreet tasks and functions, objects can work together to provide a system its functionality. The real strength lies in the fact that objects can be easily re-used and incorporated into other projects.

Object-oriented technology allows for easier maintenance. As long as the interface is designed properly and continues to function as expected, changes in the inner workings of an object will not affect the system. It is only when object interfaces need to be changed that developers run into the same problems that developers of procedural code have -- that of finding all references to the changed interface throughout the system.

Objects also isolate functionality, making it easier to track down the source of problems. Objects are easily extended, and extensions can be made in one place, often at the super class level, providing the extended functionality to all its subclasses through inheritance. And since these changes can be made in one place, adding functionality to the whole system is easily done.

Henry and Humphrey (1990) conducted a study in which they compared the maintainability of two functionally equivalent systems as performed by a group of college senior-level software engineering students. The students were divided into two groups and given two functionally equivalent systems to maintain. One was written in a procedural language and the other in an object-oriented language. Each group performed a set of maintenance tasks on the code. One group performed the tasks first on the procedural code and then on the object-oriented code. The other group did the opposite.

The results of the experiment supported the hypothesis that object-oriented code is more maintainable. It found that the object-oriented code required fewer modules to be edited, fewer sections to be edited, fewer lines of code to be changed and fewer new lines needed to be added. The authors concluded that the object-oriented code creates the need for fewer changes and that those changes were more localized and easier to do. They noted that the students who had a set of skills that biased them towards the procedural method of programming had little object-oriented programming experience, lending further credence to the maintainability of the object-oriented code.

Mancl and Havanas (1990) conducted a study of the impact of C++, an object-oriented language, on software maintenance. They studied a medium sized software system they called CXR. CXR is implemented in both C and C++, enabling them to examine the maintainability of both types of code. Their study found that software reuse doubled when object-oriented code was used, new features were added to the object-oriented sections with less effort and fewer system interface changes, and redesigning non-object-oriented sections of the system should result in lower maintenance costs. During the course of the study, they also found that any changes that required alterations to the interfaces between system modules were the most difficult to make, supporting the argument for good encapsulation of code and good object interface design.

Lewis, et al., (1991) conducted an experiment in which college senior level software engineering students were tasked with building a specified target system. The tasks given to the students involved data management, numerical processing, and graphics. The students were divided into two groups: one received a set of reusable object-oriented components that had been constructed specifically for the study and was encouraged to reuse code; the other group was given no code modules and discouraged from reusing code. The productivity of the two groups was measured. The authors concluded that the object-oriented paradigm employing code reuse substantially improved the productivity of developers, and that the object-oriented paradigm makes code reuse significantly easier.

Stark (1993) concludes that object-oriented technology is the most significant technology ever studied at the Software Engineering Lab, part of the Goddard Space Flight Center. The lab has studied numerous technologies that affect the development of software. They concluded that object-oriented technology promotes code reuse and eases the reconfiguration of developed systems. Since object-oriented technology covers the entire spectrum of development, it can positively affect the whole life-cycle of a project.

Currently, object-oriented technology is becoming the dominant paradigm in the development world. Virtually all development tools are moving toward an object-oriented framework. Even the developers of the Ada language have recently released the specification for Ada95, which includes object-oriented extensions. It is difficult to find a commercial development package that does not claim to use object-oriented technology in some way.

Object-oriented programming techniques can be shown to improve programmer productivity through code reuse and better maintenance techniques. Clearly, organizations both within the DoD and without are seeing the benefits provided by object-oriented technology.

3. Object-oriented Technology in the DoD

Diskin et. al., conducted a study sponsored by the Defense Information Systems Agency (DISA) that examined the potential use of object-oriented technology by the DoD. The report looked at the concepts of object-oriented technology and how it might fit into the various information technology initiatives and programs currently in place within the DoD. The report drew the following conclusions:

- Support the introduction of object-oriented technology to its software engineering strategy
- Develop a technology transition strategy for the introduction of object-oriented technology
- Reevaluate the process model for IM engineering
- Do not mandate object-oriented technology as an approach for all software development nor adopt any specific object-oriented methodology or technique at this time.
- Make object-oriented technology concepts part of Ada and software training
- Support efforts toward a commonly accepted definition of object-oriented technology
- Investigate the feasibility of extending the current DSRS to more fully support object/class search and retrieval.
- Encourage the development of object-oriented database management system bindings for Ada9X
- Develop a standard set of Ada class libraries
- Support ongoing and new research in object-oriented technology metrics

The report points out that a large portion of the DoD's budget is dedicated to maintenance, and that anything that can help reduce maintenance costs would certainly be of interest to the DoD. In addition, it points out that the use of object-oriented technology can increase and enhance code reuse, further stretching the DoD's IT budget. Though the benefits of code reuse through object-oriented technology have not been fully realized, the study mentions that the potential exists, through the use of inheritable class libraries, for tremendous gains in code reuse.

The report's main shortcoming is that it failed to realize that object-oriented technology cannot simply be placed inside the current initiatives. Object-oriented technology itself, like visual tools, is a radical departure from previous development methods and nothing short of a complete methodology shift will achieve its full benefits. In addition, the report recommends a number of enhancements to the Ada language which, while moving in the right direction, would continue to leave Ada behind more current, market-driven visual development environments. Commercial visual development tools already have extensive class libraries and straightforward connections to DBMSs that Ada does not have. The development of such extensions to Ada would be a waste of resources in light of tools currently available. Trying to press Ada into the object-oriented realm, even with the advent of Ada95, will likely repeat the failures of Ada83.

A visual tool that takes advantage of object-oriented technology provides developers with all of the advantages described above. Visual tools will normally encapsulate the standard interface controls into objects, allowing for their easy enhancement and extensibility. Object-oriented tools allow developers to develop custom controls for specific needs. They may even provide an object-oriented wrapper around database controls, which would enable developers straightforward access to data tables and provide them with easy ways to design interfaces that display that data. And by allowing developers to build data structures that are objects, they can reap all of the benefits that object-oriented technology provides in their natively built code.

IV. VISUAL DEVELOPMENT METHODOLOGY

A. INTRODUCTION

Visual development tools afford a tremendous opportunity to a project manager willing to utilize them. However, managers utilizing visual tools need to be careful not to merely try and make them fit into the old paradigm. Simply squeezing a visual tool into the structured design methodology will not provide a good fit. To be used to their maximum potential, such tools require a completely new way of developing applications. The development process must be completely re-engineered to take advantage of the technology and capabilities that visual tools provide. Just as structured development tools drove programmers to a structured design methodology , so visual development tools drive them to a visual design methodology. However, employing a visual design methodology requires a complete re-engineering of the process from its current state.

McDermitt (1995) has already put forth a development methodology for client/server applications that employs the Rapid Application Methodology and visual tools. The visual development methodology presented here attempts to specify the exact process during the implementation phase of such a methodology, and to describe how visual tools specifically might be used in the implementation of such systems. Emery and Zweig (1993) discuss what they call a "new paradigm" for software development that promises large programmer productivity increases, an iterative life cycle that allows for adaption in the development process, shorter, more controlled development cycles with smaller development teams, and overall reduced costs and improved quality. The Visual Development Methodology attempts to describe a specific way of employing this 'new paradigm'.

Winograd (1995) believes that developers need to shift away "from a constructor's-eye-view to a designer's-eye-view, a view that takes the system, the users, and the situation of use all together as a starting point." He argues that the experience of the people who use software should be a more important consideration that what the software actually does. What people do with the software and the environment in which they are found should become the important design criteria. He submits that the ability of tools such as visual tools to provide almost interactive development gives developers the ability to almost "converse" with the application -- developing, changing and even tossing the application out and starting over. Being able to interface with the application itself, vice the mere abstract representation of the project, makes for a more intuitive way to design and communicate.

A methodology that embraces these concepts will go a long way towards improving the quality of applications that it produces. As discussed above, visual tools allow developers to quickly make changes to applications and adapt them to changing business rules and user requirements. The fact that such changes can be made alters the way that developers should attack a problem. Rather than freezing a system with a set of written specifications that define a very specific system, developers should embrace a new methodology that provides for interactive development cycles and an ever evolving application. The methodology needs to provide developers with the ability to change and adapt the application faster than the business cycle is changing. In doing so they can stay ahead of, or at least keep up with, users needs. Any software application is really a static representation of a dynamic environment. The visual development methodology is an attempt to make a software application more dynamic and responsive to its environment.

B. THE VISUAL DEVELOPMENT METHODOLOGY

The visual design methodology is outlined in Figure 3. This development methodology encourages the developers to provide a basic application, which is then followed by incremental improvements and enhancements as users exercise the system, finding flaws, areas for improvement, and ideas for enhancement. It allows for changes to a system in a relatively responsive manner and it provides a means of allowing the business rules to drive the system, and not have the system drive the rules.

The methodology begins traditionally, with the gathering of user requirements. However, the extent and thoroughness of this examination will be very different from the traditional approach. Rather than exerting a large amount of effort and expending a sizable

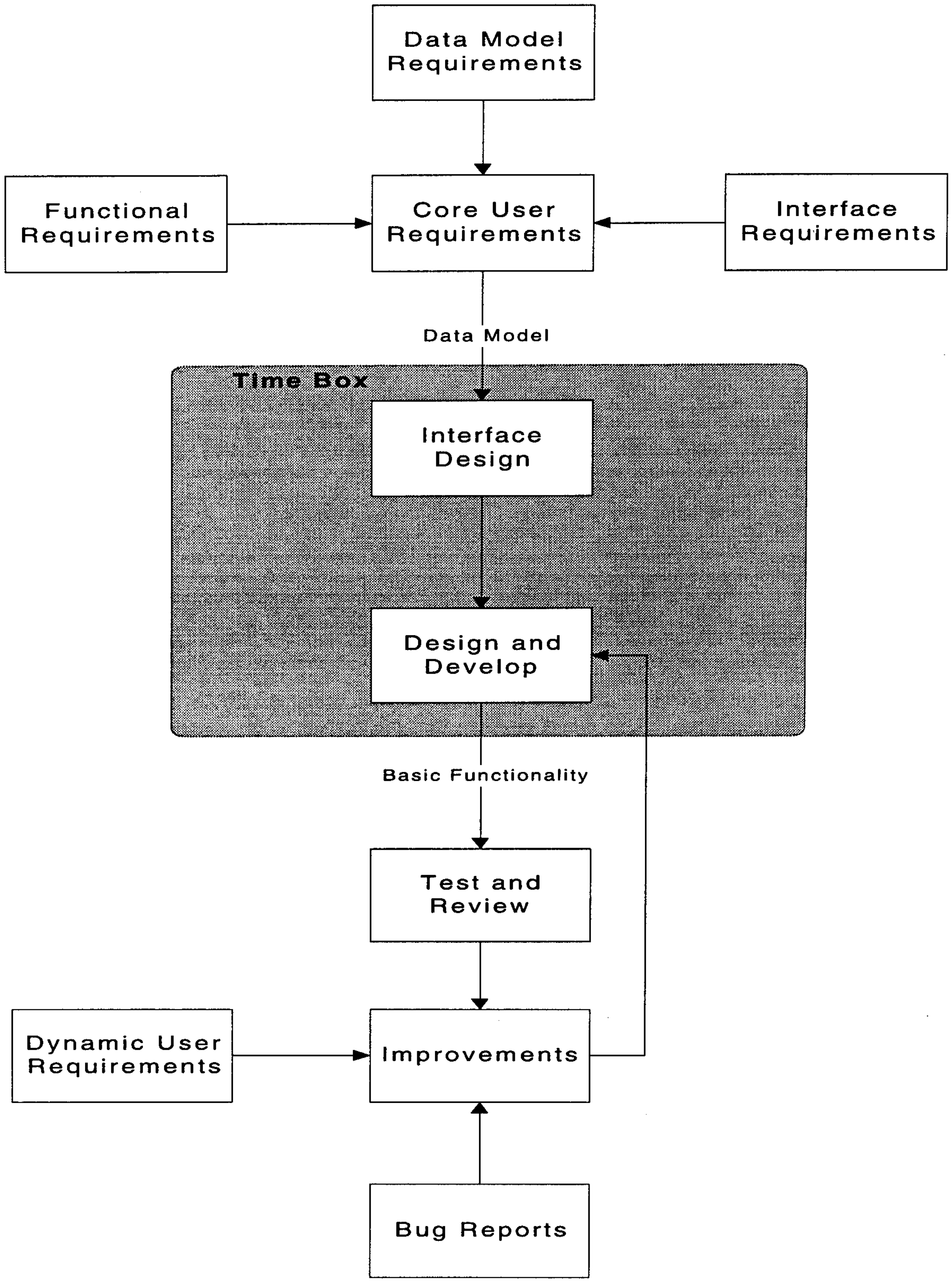

Figure 3 The Visual Development Methodology

portion of the development budget in an attempt to do an exhaustive investigation of the requirements, developers should concentrate only on the core functionality of the system to be modeled. This will allow them to begin immediately constructing an early version of the application. They can design the basic data models and object models in such a way to allow for extension. Developers should cooperate closely with users to determine what the basic version of the application should do. They should examine the business processes being modeled and determine the basic, core function that the new system is to provide. Focus can then be placed on building a system that provides this core functionality. Attention should be paid to the processes that are not anticipated to change.

At this point, and indeed all throughout the process, users can have a strong input into how the application will look and operate, as this will be a big factor in the application's acceptance and the productivity level of users as they employ the system. Getting the interface right at the outset will save coding changes later, though visual tools will make any interface changes very easy. Developers at this point should actually be building what will amount to a skeleton for the actual system, presenting users with input and output screens, report views, and other interface elements. Visual tools will allow them to make changes, often right in the presence of users, to ensure that the interface design is satisfactory and meets the needs and desires of those to be using the system.

While user input is very important to the process, it certainly should not dictate how the application is developed. McKeen et al., (1994) discuss the role of user input and the contingency factors that determine user satisfaction. The input of the developers and their decisions about user inputs should be the real driving force behind design decisions . User inputs are important, but they must be balanced with the knowledge and analysis provided by developers (McDaniel et al. 1994). Figure 4 shows how both users and developers have a say in how the application develops. Programmers will have ideas and ways of doing things that may appeal to users but never would have occurred to them. Users may have ideas and ways of doing things that developers would never realize. The combination of expert developers who know the business of building applications and users who know the

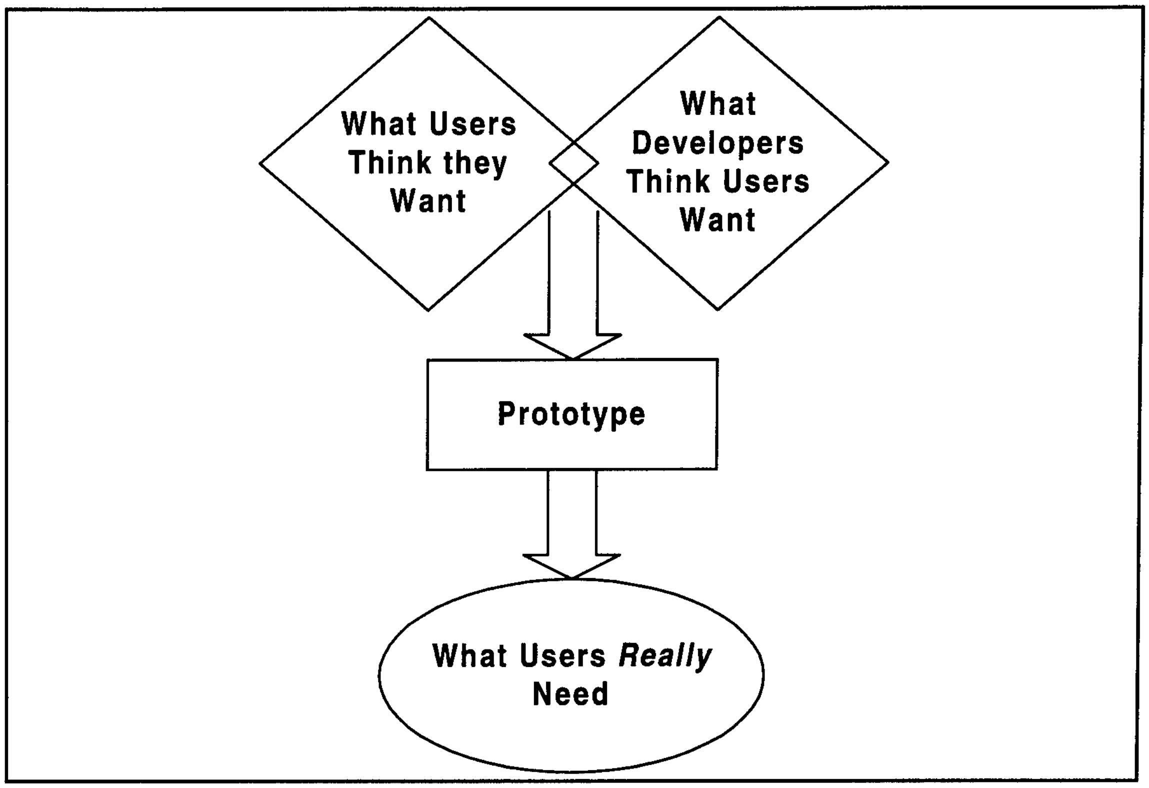

Figure 4 User and Developer Inputs to the Development Process (Comaford 1995)

business being modeled can combine to make a very powerful and useful application if both are given due representation.

During these early stages, developers should emphasize the data model -- the one area in the new visual methodology that needs to be correct from the outset. The data model first and foremost must accurately represent the actual business situations, conforming to the strict definitions of the relational model. But just as importantly, it must be designed with an architecture that allows for expansion and adaptation. Such a design needs to be built in from the ground up, and designers need to be aware of the fact that the data model will no doubt be dynamic and expanding as time goes on.

In addition, of course, developers must gather the traditional information about the functions and workings of the system to be modeled. Developers should take care, however, to not merely pave the cow paths, but rather to try to define those functions in terms of their underlying objectives and not merely automate a bad process. Developers and users, working together, can look for ways to develop a better way of doing business as a result of the technology available to them. The beauty of this development methodology is that once the thought process has begun, the foundation for a new path can be laid, and the application can adapt to the new ideas and concepts as they are developed.

At some point, then, developers will begin to gain a good grasp of these core capabilities needed by the application being developed. Developers should begin building the final interfaces and the core functionality and logic for the basic functions of the application. Developers should be very conscious not to try and develop the whole system at this point. Rather, they should concentrate on delivering a product that contains the basic functionality of the system. It should be viewed as a starting point from which further development will take place. These early versions of the application give developers and users a common reference point for further development. (Comaford 1995)

Once the ball is rolling towards this basic version of the application, developers should be placed into what is often referred to as a "time box." A time box is a limit on programmers that places a deadline on the production of a the baseline application. Developers are told that they have a fixed amount of time to produce the application, and what ever level of functionality has been completed at that time is what is initially delivered. Time boxes have the effect of compressing the development cycle and providing a development team with a goal to meet rather than the potential of a project that drags on seemingly without end. It also has the effect of providing the necessary pressure to enhance productivity and produce at least minimal results. Research shows that programmer productivity gains can be realized for certain relatively short periods of time if they are feeling pressure, either from a manager or from a deadline (Abdel-Hamid and Madnick 1990). Time boxes take advantage of this phenomenon by providing a deadline while allowing developers to know that only limited functionality need be implemented. Once the deadline has arrived, a development team should deliver an application with core functionalities.

This application can then be installed and employed by end users. The core functionality alone should provide the users with some capability that can prove useful to their jobs. At this point, they can evaluate the application, uncover defects, and even use that application as much as they need or want to. The can provide feedback as to the success or failure of the design decisions made during the requirements analysis. Getting the application into users hands as early in the development cycle as possible will allow developers to make corrections and fix defects early in the development cycle rather than late, when they are far more costly to deal with. Because users are actually able to deploy the system with its base functionality, they can provide an evaluation as to the suitability of the application for real world use. Using the application will enable users to formulate the direction of the application in ways that they may not have thought of until actually using the application.

Once the initial version of the application has been deployed, developers begin the process again. They should solicit and accept user inputs, now in the form of bug reports, ideas for enhancement, and ideas for new functionality. Once users are actually using the application, they will have a much better idea what the application will need to do. The project manager must gather and prioritize the feedback, mixing his or her own expertise with the requests and knowledge of the users. The manager then can determine how the application will be improved and implement those features are determined to be of the highest priority. Again, after another time box has been established and the next version has been completed, the new version of the application can be deployed, and the cycle starts over again.

C. DISCUSSION

The visual development methodology represents large change from the traditional approach using structured analysis and design. The largest and perhaps most difficult shift in thinking is in accepting the idea that requirements specifications are no longer done solely up front. In the traditional approach, a complete an exhaustive requirements definition is considered essential to a good product. One would not even think about implementing a system without a good requirements definition. Great efforts and large amounts of resources are expended to produce these requirements. Frequently, however, these requirements are often overtaken by events as the business rules change. Over time, they cease to describe the system needed by the end users.

The visual development methodology takes a different approach. Instead of doing a complete requirements analysis at the beginning of a project, the methodology recognizes that those requirements are going to grow, evolve, and change over the life of the application. Instead, developers seek to build the very core functions of the application, using only a basic requirements definition. In avoiding this large effort, the adaptive methodology lowers costs by not doing work that will merely be superseded, while allowing the application to grow and adapt to the changing work environment. The process becomes more efficient by not doing work that will later be irrelevant. Design changes become the norm rather than the exception.

Such design changes and ideas can be made immediately as the project develops. User inputs provide direction for the development team to continue to build the application from its base functionality. The use of visual tools enables developers to make these changes and enhancements as they are made aware of them. Thus, the application can actually evolve in a direction that suits the needs of users. Developers, rather than avoiding such inputs, use them as their main source of design and implementation direction. Developers should never have an idea in their mind about what the application should look like. They should instead realize that the goal of ending the project -- at the point that it will actually be 'finished' -- is not one that may ever be cleanly attained. Rather, they should view themselves as maintaining and evolving the application in a direction that they see, at least in the long term.

As iterative changes are made, developers can provide users with another interim version of the application that incorporates the fixes, enhancements, and changes that users have recommended. As updates are provided, users see how their suggestions have been implemented. Each iteration would provide them with an application that is more functional.

Users can watch the evolution taking place.

Eventually, they will find that the application has become ready for full-scale, enterprise-wide use. Seeing the application evolve towards one that they can use effectively will motivate users to become even more involved in the development process and can stimulate them to make more and better suggestions for improvement. With each iteration, as it grows closer to a complete, enterprise system, the application will get more and more real-world use, revealing defects and gaps in the program. Users will not only begin to see how technology might alter their business rules, but as the application adapts to meet the needs created by those changes, a positive cycle occurs. As users see an improved application -- which improves their productivity and efficiency -- they are stimulated to come up with more and better ideas for improvement. Developers working closely with users will have a better idea of how the users work and what rules govern their work. Developers are able to meet these needs and implement changes as they occur.

This deploy-evaluate-design-deploy loop is a radical departure from previous methodologies. Historically, development projects had clear milestones that could be defined and measured, and it was clear when the project was over and when the maintenance phase begins. There were clear stopping points. With visual design, such clean phases are not easily identified. It is not clear when an application is 'done,' as the application is more evolving than it is being written. The process does not really have an ending point. The development team may shrink from its original size, but the application will continue to change as the needs of users change. One no longer needs to define an implementation phase and a maintenance phase. The distinction between the two is not necessary, and it is really impossible to say at what point one stops and the other begins.

Thus, in looking at this methodology, it becomes clear that the concept of phases becomes very blurred. The difference between specification development and implementation, or the difference between implementation and maintenance, become harder to see because the building of the application pervades virtually every area of the entire development cycle. As shown above, development will begin in the very early stages of investigating the problem to be solved. Saying that development has come to a close and the application is now 'done' may even prove problematical. Instead, a project will continue on indefinitely. As long as there is a need for the application, developers should continue working on it. What was maintenance now becomes implementation.

This new methodology, therefore, means a redefining of the meaning of the maintenance phase. Historically, maintenance, or the upkeep and improvement of the program after it has been deployed, has been the largest portion of a projects total life cycle costs -- often as much as 80 percent of these costs. However, the name maintenance can be misleading. Often, maintainers are not so much called on to fix errors and bugs in the code as they are to adapt the application to the changing needs of users. What is called maintenance is often really reworking the application to meet the current needs of users. Maintainers end up doing what should have been done during the development cycle.

Swanson and Beath (1989) support this point. They discuss the organizational and managerial problems associated with the maintenance phase of development. They point out that the very concept of maintenance is unclear, and that majority of maintenance time is spent improving and adapting currently installed systems. Rather than persisting in treating maintenance as a second-class process, the authors argue that maintenance should become part of the overall development cycle. Instead of drawing a line between new development and maintenance of current systems, managers should differentiate between installed systems and future systems. Thus, maintenance ceases to be an afterthought and becomes part of the original development process itself. This position is in keeping with the visual development methodology, which does not draw a line between maintenance and development.

During the traditional development process, great effort is put into eliminating errors and bugs that creep into code as the project progresses. Errors found early in the process are much cheaper to fix than those found later in the project. Development managers take great pains and expend many man-hours to do quality assurance checks, code reviews, and testing to ensure that the code performs as advertised. That is all well and good, but if the end result does not meet the current user requirements, then the maintenance portion of the life cycle will involve rewriting the application to meet current user requirements.

The visual methodology accounts for this by putting a larger emphasis on ensuring that the application meets current user needs and less emphasis on the initial specifications or delivering error-free code. That is not to say that errors are acceptable, but rather that they are more acceptable than delivering an error-free project that is of no use to the user. As a result, maintenance costs can be saved by realizing time savings throughout the entire process, and not leaving the real, needed enhancements until the end.

However, any changes and improvements that are made after an application developed with the traditional approach has been deployed are much more expensive to do than if they had been done as part of the development cycle. In addition, trying to adapt an application after it has been completed, late in its life cycle, is far more costly, difficult, and error prone. It is always better to get it right the first time, whether talking about errors or functionality. The visual development methodology allows for the 'maintenance' of a project to begin very early in the development cycle. Projects should be viewed as being in the maintenance mode right from the beginning, rather than at the end, of the project's life cycle. Visual development will also mean a large departure from the traditional role of the software engineer. In the past, the software engineer has been tasked with completing a project. This was the measure of success or failure. Failure is not defined as providing a system that users need, but as meeting the goals set at the beginning of the project. Delivering a useful project to users is not a motivating factor. Merely meeting requirements is. A major problem in the DoD is that the control procedures generally penalize developers that change specifications to meet real user needs.

Projects are normally budgeted with a specific cost of development in mind, and once the project is done, they are budgeted for maintenance. Managers are pressured and work very hard to meet such budgets. They are not rewarded so much as for succeeding as for not failing. As seen above, projects rarely meet these budgets. It should be realized that such budgets are most often unrealistic.

The visual development methodology changes this. Software managers should be more interested in providing a working product rather than merely a finished product. The visual development methodology allows them to concentrate their efforts on meeting users needs rather than schedules. Costs can be contained through an iterative process that provides a steady level of effort. Time boxes will allow for a more controlled development environment. The incentives inherent in the system will cause managers to produce applications that conform to the needs of the users, and not merely to a frozen, perhaps meaningless, specification.

The visual development methodology is a means of properly using visual development tools. By allowing for rapid interface development, iterative application delivery, and user inputs throughout the whole process, it can deliver the long hoped for productivity gains for development teams.

V. PROBLEMS WITH THE VISUAL DEVELOPMENT METHODOLOGY

A. WEAKNESSES IN THE VISUAL DEVELOPMENT METHODOLOGY

Clearly the biggest weakness with the visual development methodology is that it is largely untested. Some organizations have approached this technique using Rapid Application Development with great success (Hodges 1994; Emery 1991). Visual tools are a relatively new advancement in programming tools and, as such, their effectiveness is not yet fully known or realized. The market for visual tools has grown rapidly, and certainly private industry has been quick to embrace these development tools. As the use of such tools increases, organizations will begin to look for better ways to employ them. Methodologies such as the one put forth here will begin to come into use. Perhaps soon there will be a larger body of knowledge about their usefulness. Such information will no doubt prove useful to the DoD as it evaluates these new methodologies. However, until that happens, the visual development methodology will remain an unknown entity.

In addition, the outcomes of visual design and object-oriented programming are very difficult to measure. Managers need to be able to measure accurately any process that they are expected to control. There are many ways currently in use to measure the length, cost, and completeness of projects developed under the structure design methodology. However, established estimates based on lines of code, COCOMO models, and Function Points cease to have much meaning in object-oriented and visual development environments.

Currently, there is much study being done in the area of measuring object-oriented systems. When visual design tools provide much of the code in a project through code generation, it is hard to measure productivity using the methods from structured design. Lines of code is meaningless when dealing with code that makes heavy use of inheritance. Many visual tools allow the development of basic database applications without a programmer writing a single line of code, so the traditional methods of programmer productivity and project completion no longer apply. Tools to measure the visual development process will be particularly important in the early stages of its adoption as managers try to determine if it is indeed a more productive and effective methodology. Chidamber and Kemerer (1994) have put forth a suite of metrics for measuring and quantifying the complexity of object-oriented designs. They also provide an excellent summary of research done in the area up to this point. Further study in this are will no doubt reveal useful methods for measuring such systems. But until such time as managers have useful tools available to them, the visual design methodology will be difficult to measure and quantify.

Version control and distribution of product versions will also prove to be a problem. The visual development methodology involves the frequent distribution of updated applications. The process of controlling versions across a wide enterprise and ensuring that all users have the most current version available will be difficult tasks for a manager to oversee. In addition, constant iterative updating of the application can become a "runaway freight train," with the users driving the development process rather than following some organized, structured plan for development. There is a danger that user suggestions could take over and usurp sound project management (McDaniel et al. 1994). Controlling the development process, both in terms of version control and design decisions, will require excellent managerial control.

B. IMPLEMENTATION STRATEGIES

While the visual development methodology may be an effective and powerful means of developing better software more efficiently, adapting it will no doubt prove problematical. It represents a complete break with the current methodology in use by the DoD. It is also untested with the large, enterprise-wide projects that the DoD frequently undertakes. Deploying the visual development methodology will require careful study and close management for it to be successfully integrated into the DoD.

Instituting the visual design methodology will have to begin slowly, with test cases at the low levels of the DoD. Commanders as low in the chain of command as air wings and ships should be encouraged to use the methodology to develop systems to meet IT needs at their level. The DoD could institute pilot programs to study and monitor the success levels of such projects. As the methodology is more fully understood and its process becomes accepted, the methodology could be pushed higher into the organization. Gradually, larger and more important applications could be built using the methodology.

C. DIFFICULTIES WITH CONTRACTING FOR SOFTWARE DEVELOPMENT

Current laws and instructions concerning the development of software will make the adoption of the visual development methodology very difficult. Current DoD contracting methods require extremely specific and clear-cut specifications for any money that the DoD spends. Historically, contracts for software development have been very specific in the performance, functionality, speed, and capabilities of the software product to be built. Structured analysis and design is practically written into the contract, as DoD contracts contain specific milestone agreements and stages that the contractor must meet. The contracting process allows little leeway for deviating from the contract. The rigid testing requirements and traceability analyses that are required by DoD contracts generally require strict adherence to the initial specifications.

Trying to match these laws with the visual development methodology will be difficult. Projects will have to be viewed as going into what is presently known as the maintenance phase almost from the start. Instead of clear-cut steps towards a specific goal, software contracts will have to be viewed as long-term support agreements in which the vendor agrees to provide continuous service.

Software development contracts will have to be viewed more as a service contract rather than a 'goods for fee' type contract. The visual development methodology will demand a more maintenance-oriented approach to software development. The government will no longer be buying a product as much as contracting for a service. Current contracting procedures require strict competition in bidding for systems. Current contracting law makes service contracts available for re-bid after limited amounts of time. Changing contractors in the middle of a project will not allow for the continuous updating needed for the Visual Development Methodology to work. Software development contracts will be difficult to administer in this environment because current contracting law discourages, indeed almost makes it illegal, for the government to develop long-term relationships with software vendors. For this new methodology to be put into place, the government's aversion to long-term relationships with vendors will have to be overcome.

D. CULTURAL CONFLICTS

Perhaps the most difficult change of all is the large cultural shift that will have to take place if the visual development methodology is to take root in the DoD. Years of doing business in a certain way will not easily yield to a new way of thinking. The DoD has been attacking the problem of getting its software development process under control in a certain manner; radically moving from its current method of close control and strict monitoring to a system that asks for less control and monitoring will not be readily adopted.

The DoD's current response to its difficulties has been to exert more control and regulation over the development process. When the Government Accounting Office was asked to look at the problem, its response was, in effect, that the DoD was not exerting enough control and not enforcing its current rules and regulations forcefully enough. Instead of looking at the problems inherent in the system itself, the DoD has preferred to establish agencies to monitor and control the software development process, further burdening developers with bureaucratic requirements (GAO/IMTEC-90-36).

In the late 1970's, realizing that a large number of software development projects had gone over budget and schedule, the DoD created the Major Automated Information System Review Council (MAISRC). It was commissioned to oversee the requirements justification, design, development, and deployment of major AIS developments and acquisitions. The MAISRC was intended to provide oversight and provide close scrutiny to major development projects at the key milestones to the projects. In addition to milestone reviews, the MAISRC is authorized to hold in-process reviews, conducting inspections in between milestones with the purpose of further reducing risks. MAISRC is an attempt to inject accountability into the software development process in the DoD and to gain a measure of control over runaway budgets and undelivered functionality. (House Report 101-382)

The MAISRC may have achieved those purposes, but what it also did was increase the costs and the amount of time that software projects take to finish. By adding another layer of bureaucracy in an already overburdened system, the effect of MAISRC and other control measures like it is to actually make the process worse. If project managers have to spend time preparing for MAISRC inspections, they cannot spend time managing and developing. Capers Jones reports that the DoD requires approximately 400 English words of documentation for every line of code written (Yourdan 1994). Time spent away from the task of developing the application is time wasted. Rather than focusing on more strict controls, the DoD should focus its effort on improving the process itself. Embracing a methodology that uses visual design tools would be a step in the right direction.

For the DoD to successfully change its current pattern of wasteful, high-cost software projects, it will have to take a completely different view from the authoritarian response it has been using. This is likely to be a difficult change in attitude. In 1991 the DoD held a series of discussions with various personnel involved in information resource management in the DoD, including GAO staff, IRM officials, and private sector representatives. They discussed a wide range of issues concerning the troubles in the DoD. (GAO/IMTEC-92-67) They came up with a number of problem areas and recommendations for improvement, but it is interesting to note that none of the comments or recommendations mentioned the possibility of a fundamental flaw in the current system. All discussion centered around ways to fix or improve the present way of doing business. A common theme to the discussions seemed to be that the process would improve if managers could merely find ways to get better control of it. These panel discussions make clear that the current system is deeply entrenched and that any effort to alter the current methodology is going to meet with serious skepticism and resistance.

The fact that the visual development methodology has weaknesses and will be difficult to implement should not deter DoD managers from implementing it. The current system is clearly flawed and is costing taxpayers billions of dollars each year. The DoD has attempted to improve the process by putting in place various programs that promise incremental improvements at best. Instead, the DoD should focus on implementing methodologies that will provide revolutionary, not merely evolutionary, change.

VI. CASE STUDY - THE PERVIEW SYSTEM

A. INTRODUCTION

The majority of this chapter contains a report written by LT Doug Swanson, USN, LT Todd Barnum, USN, LT Nick Hodges, USN, and LT Lee Stone, USCG, for the IS 4925-3 class entitles "Client/Server Application Development" during the Summer 1995 quarter at the Naval Postgraduate School. It is the report submitted concerning a class project that dealt with the development of a client/server application. This chapter does contain material that examines the issue of the use of the visual development methodology more closely than did the original report. This chapter serves here as a case study of the capabilities of the visual design methodology.

B. PROJECT BACKGROUND AND DESCRIPTION

1. Background

Currently, the Naval Postgraduate School (NPS) has no centralized database from which to generate class rosters with detailed information about students. Rather, class rosters are periodically made available in hard copy. Throughout the school there are multiple databases maintained by different departments employing a variety of applications and computer platforms. Specifically, class information and rosters are maintained by the registrar's office on a FOCUS database, while detailed student and faculty information is either dispersed throughout the school or does not exist at all. Clearly, there is no standardization regarding the data model and the information that should be captured for students or faculty.

2. Application Description

The purpose of this project was to develop a system to provide detailed, flexible access to class roster information on-line. The NPS faculty members were viewed as the primary clients. Curriculum offices, the Superintendent's staff, and other administrators would undoubtedly find the application useful as well.

The application was developed using Borland International's Delphi visual development tool. During project development, one principle was considered paramount: the application would utilize existing databases rather than create yet another non-standard, stovepipe database. With that in mind, the application uses data from the FOCUS database for class roster information, and a simulated Oracle server for the detailed student information. The Oracle server is currently being installed on a shared database server, and was simulated in the application by means of Delphi's local Interbase server. The FOCUS database uses a simple script to dump a comma-delimited ASCII file to a mainframe account. The data can then be retrieved to the local PC and mapped to any form for use in the application. Since the data was modeled using SALSA, a semantic object data modeling tool, Paradox data files were originally used, but the data was later transferred to employ the Local Interbase Server as the database management system. Faculty information could be obtained from a variety of sources including the Faculty On-Line Database Resume system (FOLDeR). For the purposes of the prototype, the faculty data is also simulated by the Interbase server.

The application itself consists of a simple toolbar that gives access to different views of the data. The views are class, student, and faculty. While not specifically required, faculty was added because a faculty view logically fits with this type of application. Once a view is selected, certain data elements can be selected to provide greater detail. For instance, when viewing class information, a student name can be double clicked to see additional student information. Detail information includes a photograph and a list of other classes in which the student is currently enrolled. While simple and intuitive to use, the application does provide on-line help.

C. BENEFITS

The benefits from such an application stem primarily from the increased functionality afforded to the faculty and administrative staff. This application will allow any professor or staff member to gain immediate access to course roster information. Additional detail information can be obtained on both faculty and students. These rosters will be updated as add-drop forms are input to the registrar's FOCUS database, which is then 'dumped' to the mainframe. This can be done on a daily basis. Therefore at no time will the application's information be anymore than one day later than the information provided via the registrar's office.

The additional detail information provided in the application not only provides faculty members with name-to-face recognition, but provides other unexpected benefits as well. For example, by clicking on a student name, the student's entire schedule is available. This essentially renders the current paper system of student locator cards obsolete.

Possibly the most significant benefit of this application is that it demonstrates the capability to link existing databases to gain an enterprise view of the data. This can be done without creating additional stovepipe databases.

Determining the associated cost savings from the proposed application was beyond the scope of our project. The project team recommends that a cost-benefit analysis be performed as a follow-on project so that the true value of the system can be ascertained. It should be noted that the benefits of the application are largely improved functionality rather than operational cost savings.

D. USERS MANUAL

This application is relatively simple to use. The two disk set can easily be installed by simply running the setup program from Windows and following the on screen directions. Once launched, the program is self-explanatory. On-line help is available should any questions arise.

Updating the data from the FOCUS database is somewhat less intuitive. Ideally, this step would be automated, but the time limitations prevented that in the prototype. A two step process is involved in updating the data. First, the data must be retrieved from the mainframe. This is done by using an FTP tool to access account 5957p. The read-only password is @pvw2. Second, the data must be mapped to the correct format. This is done by a small Delphi utility program that allows for transfer of data from the comma-delimited ASCII format to Per-View's native data base format (Interbase).

E. DEVELOPMENT METHODOLOGY

The methodology employed to develop this application was the visual development methodology. This approach uses a visual object-oriented application development tool, in this case Borland's Delphi, to rapidly develop an application prototype. Borland's Delphi is a development tool specifically designed to create client-server applications.

This type of tool allows the developers to build highly sophisticated applications with very little code writing. These products provide the developer with a library of pre-packaged modules that can be visually combined into complete applications. The real power of these tools stems from this library of components that allow an application to be assembled with connections to databases, video, imaging, and messaging.

The purpose in using a visual tool is to provide the user with a prototype as quickly as possible. The user can then play a significant role in the applications development cycle by continually providing the programmer with feedback. This iterative process is commonly referred to as RAD and promises to greatly improve the software development process that for years has been stymied by rigid methodologies.

In the case of this application, requirements were identified, and a logical schema developed. Figure 5 illustrates the schema used. Briefly, the data model consists of a 'Person' supertype, and subtypes of 'Student' and 'Faculty'. This data will presumably be provided by accessing other databases currently under development, but is currently simulated using Delphi's local Interbase server. The 'Class' table provides information on each class, and the two intersection tables detail which students and which professors are in each class. The information in these last three tables is obtained from the FOCUS database.

Once the schema was developed, work on the application itself began. Concurrently, the method for accessing the FOCUS data was developed. Significant time was saved by doing these two simultaneously. Once the FOCUS script was functional, it was relatively

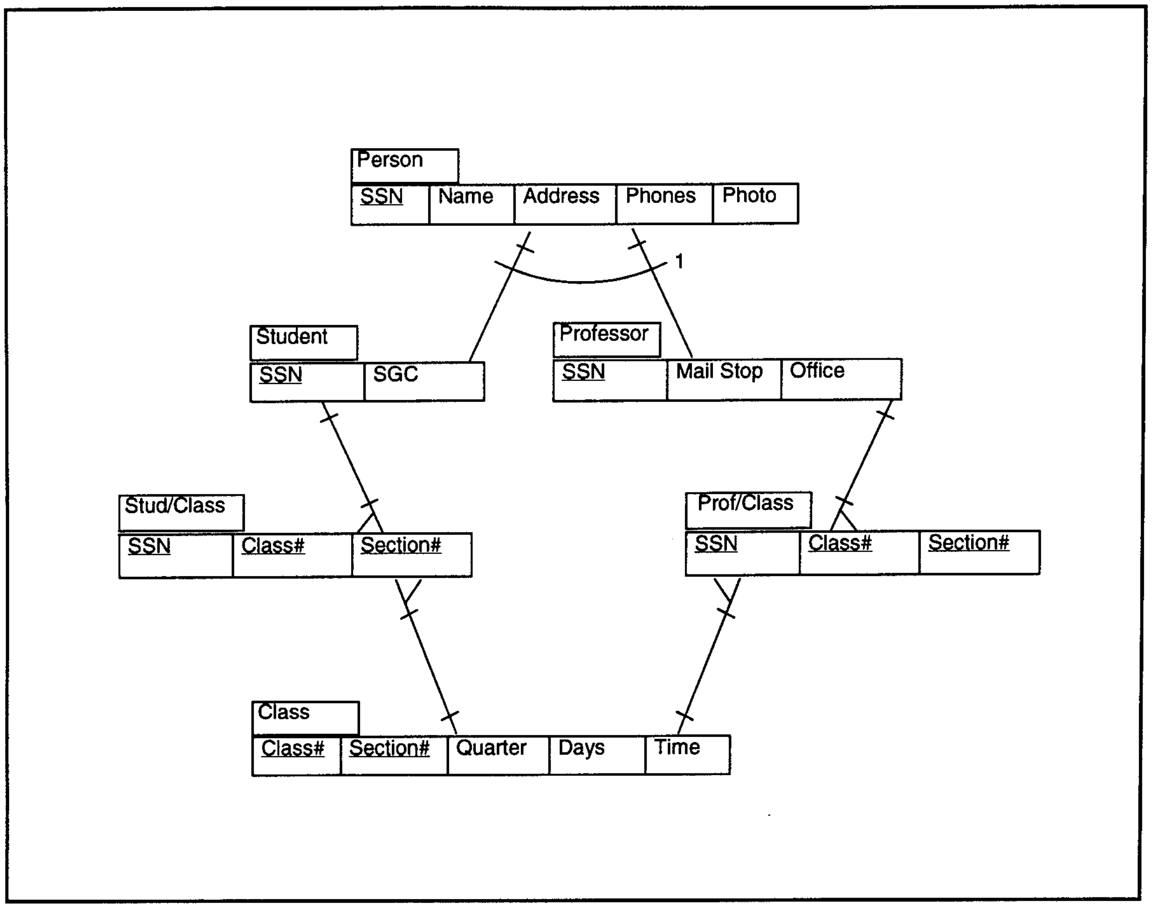

Figure 5 Perview Database Schema

easy to FTP the files from the mainframe, and re-map them to the application format, which is Interbase.

F. APPLICATION ASSESSMENT

The quality of our application is a testimony to the power of such visual programming tools as Delphi. Designing an application of equal functionality without the use of a visual programming tool would be a tremendous task. This application has been designed within the time constraints of one academic quarter.

1. Points of Contact

Tracy Hammond in the Registrar's office is the point of contact for the FOCUS data and the script that uploads the data to the mainframe account. Jim Hart of the Computer Center is the point of contact for the Oracle data. Professor Emery was the initial user for whom the application was developed.

2. Follow-on Needs

While the application is fairly complete, there are a few areas where additional work is required. First, the script which writes the FOCUS data to a mainframe account needs to be automated so that it runs at a specified time interval. Next, the application needs to provide a feature that will allow automatic update of the data from FOCUS. This could be accomplished by giving the user the option to update data as the application loads. If the user chooses to do so, the program would automatically access the mainframe, retrieve the updated ASCII files, and map them to the proper format. The team felt that this level of automation was feasible, but beyond our current time and knowledge limitations. Finally, the report features are not fully functional. This is due primarily to difficulties with the ReportSmith portion of Delphi.

G. DISCUSSION

The Per-View project illustrates the potential of the visual development methodology. A group of relatively inexperienced military officers were able, in a matter of a few weeks, build a small but fairly robust client/server application that could be immediately deployed and used by NPS faculty, yet which could easily be extended and enhanced to add greater functionality. The application delivered represented the core functionality of the system, yet it was designed from the ground up to be extensible. It would be very straightforward to add more reports, additional data views, and even to enhance and augment to data model itself in order to give a fuller picture of the needed information.

The visual tool Delphi, a Pascal-based development tool, made possible the very rapid development of a basic user interface. It also facilitated the straightforward connection to a client/server DBMS. In this case, the Local Interbase Server was used, but the system could very easily be scaled up to connect to the Oracle server that will soon come on-line at NPS. The end result was a scalable, reasonably robust client/server AIS that is useful at its initial deployment. In this regard, the Per-view system is an excellent validation of the efficacy of the visual development methodology.

VII. CONCLUSIONS AND RECOMMENDATIONS

It is clear that the DoD has great trouble managing the development of AIS applications. The development methodologies currently being used can be largely blamed for this lack of success. The proscribed use of structured analysis and design locks the DoD into a specific set of tools and a regimented, bureaucratic-ladened way of developing applications -- one that can be shown to be unable to adapt to the rapidly changing needs of its users.

The trend in the marketplace towards the use of visual development tools cannot be ignored by the DoD. The Department must begin to closely examine the potential productivity and quality gains available from these types of development platforms. The visual development methodology is an attempt to outline a way in which these tools can be employed in the most productive and useful manner. It embraces the concept of rapidly changing user needs and allows developers to make the iterative changes necessary if applications are to remain relevant and effective.

Therefore, the following recommendations are made:

- The Ada mandate should be rescinded to allow for a number of different languages and development platforms to be used in the development of DoD AIS applications. Perhaps rather than limiting all development to one language, developers could be allowed to choose from a small, approved palette of tools.
- The DoD should revoke the mandate requiring structured analysis and design and begin to implement the use of object-oriented technology throughout the development process.
- The DoD should immediately begin encouraging the use of the visual development methodology throughout its domain.
- The DoD should begin training software project managers in the uses and advantages of visual development tools and the visual development methodology.

- The DoD should modify its acquisition process to allow the use of the visual development methodology.

LIST OF REFERENCES

Abdel-Hamid, Tarek and Stuart E. Madnick, *The Dynamics of Software Development*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1990.

Chidamber, Shyam R. And Chris F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, Volume 20, Number 6, June 1994.

Comaford, Christine, "Client/Server RAD Techniques...(That Really Work)," Lecture given at the Borland Developers Conference, San Diego, CA, August 9, 1995.

Committee on Government Operations, Rep. John Conyers, Chairman , Report 101-382,"DOD Automated Information Systems Experience Runaway Costs and Years of Schedule Delays While Providing Little Capability," November 20, 1989.

Department of Defense Directive 8120.1,"Life-Cycle Management (LCM) of Automated Information Systems (AISs)", January 14, 1993.

Department of Defense Instruction 8120.2, "Automated Information System (AIS) Life-cycle Management (LCM) Process, Review, and Milestone Approval Process."

Diskin, David H., "Managing the Transition to Object-oriented Technology for the Department of Defense Information Systems", Defense Information Systems Agency, October 28, 1993.

Emery, James C., Lee L. Gemillion and Sui Mei Chai, "The Software Development Process: The Strategic Importance of Getting It Right," Working Paper, October, 1991.

Emery, James C. and Zweig, Dani, "The Use of Ada for the Implementation of Automated Information Systems Within the Department of Defense", 28 December 1993.

GAO/IMTEC-89-9, "Status, Costs, and Issues Associated With Defense's Implementation of Ada," March 1989.

GAO/IMTEC-89-21FS, "Naval Aviation Logistics Command Management Information System," February, 1989.

GAO/IMTEC-90-22, "The Personnel Concept II System Is Not Ready for Deployment," February, 1990.

GAO/IMTEC-90-36, "Defense's Oversight Process Should Be Improved," April, 1990.

GAO/IMTEC-92-67, "Perceived Barriers to Effective Information Resources Management: Results of GAO Panel Discussions," September 1993.

Hammer, Michael, "Reengineering Work: Don't Automate, Obliterate," *Harvard Business Review*, July-August 1990, pages 104 - 112.

Henry, Salloie M., and Matthew Humphrey, "A Controlled Experiment to Evaluate Maintainability of Object-Oriented Software," *Proceedings of the Conference on Software Maintenance 1990*, IEEE Computer Society Press, Los Alamitos, CA, 1990.

Hodges, Nick, "Economic Advantages of Rapid Application Development Tools," Term Paper submitted to Dr. James Emery, Naval Postgraduate School, August, 1994.

Lewis, John A, Sallie M. Henry, and Dennis G. Kafura, "An Empirical Study of the Object-Oriented Paradigm and Software Reuse,"*Conference Proceedings: Object-oriented Programming Systems, Languages, and Applications*, ACM Press, New York, NY, 1991.

Mancl, Dennis and William Havanas, "A Study of the Impact of C++ on Software Maintenance," *Proceedings of the Conference on Software Maintenance 1990*, IEEE Computer Society Press, Los Alamitos, CA, 1990.

McDaniel, Susan E., Gary M. Olson, and Judith S. Olson, "Methods in Search of Methodology -- Combining HCI and Object Orientation," *Conference Proceedings: Human Factors in Computing Systems*, Boston, Massachusetts, April 24-28, 1994.

McDermitt, David R., "A Client/Server Application Development Methodology for DoD," Masters Thesis, Naval Postgraduate School, June 1995.

McKeen, James D., Tor Guimaraes, and James C. Wetherbe, "The Relationship Between User Participation and User Satisfaction: An investigation of Four Contingency Factors," *MIS Quarterly*, Volume 18, Number 4, December 1994, page 427.

DOD-STD-2167A, February 29, 1988.

MIL-STD-498, "Software Development and Documentation", December 5, 1994.

Paulk, Mark, C., Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, "Capability Maturity Model, Version 1.1," *IEEE Software*, July 1993.

Ruston, L., M.J. Muller, and K.D. Cebulka, "Designing Software for Use by Humans, Not Machines," *Proceedings: 13th International Conference on Software Engineering*, Austin

Texas, May 13-16, 1991, IEEE Computer Society Press, Los Alamitos, CA, 1991.

Simpson, David, "Behind Good Design: The Fine Art of OO Methodology", *Client/Server Today*, March 1995, page 52.

Sprague, Jr., Ralph H. and McNurlin, Barbara C., *Information Systems Management in Practice*, Third Edition, Prentiss-Hall, Inc., Englewood Cliffs, NJ, 1993.

Stark, Mike, "Impacts of Object-oriented Technologies: Seven Years of SEL Studies," *Proceedings: Conference on Object-oriented Programming Systems, Languages, and Applications*, ACM Press, New York, NY, 1993.

Swanson, E. Burton and Cynthia M. Beath, "Reconstructing the Systems Development Organization," *MIS Quarterly*, Volume 13, Number 3, September 1989, page 293.

Winograd, Terry, "From Programming Environments to Environments for Designing," *Communications of the ACM*, Volume 38, Number 6, June 1995, pages 65-74.

Yourdon, Ed, "Logicon and the Defense Department's I-CASE Project" *Application Development Strategies: The Monthly Newsletter on CASE, Client Server Systems and Software Development Technologies*, Volume VI, Number 7, 1994.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5101	2
3.	James C. Emery Code SM/Ey Department of Systems Management Naval Postgraduate School Monterey,California 93943-5002	1
4.	Kishore Sengupta Code SM/Se Department of Systems Management Naval Postgraduate School Monterey,California 93943-5002	1
5.	Barry Frew Code SM/Fw Department of Systems Management Naval Postgraduate School Monterey,California 93943-5002	1
6.	Todd Barnum, LT, USN JAST Program Office 1745 Jefferson Davis Highway Suite 307 Arlington, Virginia 22202	1
7.	John N. Hodges, LT, USN Post Office Box 221096 Carmel, California 93922-1096	2